WHITEPAPER

Embracing Generative AI: A New Stack for the AI Era



DATASTAX

SUMMARY

Generative AI is an innovative leap forward from our existing AI technologies.

Unlike traditional AI, Gen AI harnesses both structured and unstructured data to allow applications to perform "human-like" tasks.

Gen AI provides human-like responses, empowering us to replace a significant portion of human interaction in the digital realm. This shift allows us to streamline workflows, automate processes, and offer more personalized experiences to end-users.

The purpose of this document is to tie together the latest advances in AI, cognitive science, and industry to predict what will be the defining characteristics of a future generative AI technology stack. The intended audience: technology professionals who are trying to make sense of the latest developments in AI. It is not a beginners guide to generative AI. It doesn't explain the underlying concepts of the referenced documents, so we encourage the reader to follow the links.

Evolution of the stack

The evolution of technology has been punctuated by major seismic shifts, from the internet to mobile to cloud, and, to a certain degree, social media.

Each of these shifts has sparked changes in both the application (data) plane and the control plane, subdivided into compute /storage and provisioning/monitoring. This is a useful template for analyzing the stack, and how it will evolve over time.[1]

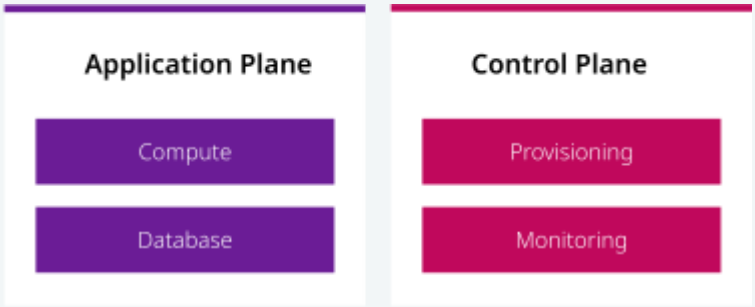


Fig 1. Division of the stack

Let's look at a brief history of how these layers of the stack have been "reinvented."



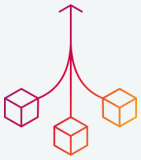
Compute

Instead of "heavy weight client & server" applications, there was a massive transition to "lightweight browser + web application servers." As a result, there was a massive boom in data center build outs.



Databases

NoSQL databases were born during the internet shift, and were accelerated by the shift to mobile.



Provisioning

Virtual machines (e.g. vSphere) underwent a massive expansion during the internet heyday, but got entirely disrupted by Amazon Web Services during the migration to cloud. This migration was very much fueled by the mobile shift, which resulted in orders of magnitude more customers and application traffic.



Monitoring

The shift to the internet sparked massive new vendors such as Splunk; many of them still dominate today. These systems were then re-invented by the shift to cloud (e.g. NewRelic / AppDynamics).

To better understand the changes likely to happen in the stack, we first need to understand the core technology shift and the new mental model for generative AI technology.

[1] https://docs.aws.amazon.com/whitepapers/latest/saas-architecture-fundamentals/control-plane-vs.-application-plane.html

From human coded applications to autonomous AI agents

The AI shift, particularly Gen AI, is poised to cause a transformation on the same magnitude as the birth of the internet. This shift is unique and fundamental because **most of the code executed will be generated by AI.**

This shift is already underway. In 2023, 92% of developers are using AI tools already for code generation (Hurler).[2] In 2022, 3% of all code (measured in characters) was also generated via AI for Google[3]. Today, Databricks has the ability to query a database using natural language[4]. Our own DataStax team was able to leverage the efficiency of ChatGPT, Co-pilot and a portfolio of AI tools to add Vector Search to Cassandra in 6 weeks.[5]

Over time, all new application-level code will be replaced with large language models (LLMs) programmed in natural language. These new applications (AI agents) will be integrated into existing applications via a microservices architecture. This will enable the gradual introduction of AI agents, especially as these agents become more powerful. We are already seeing examples of this type of integration through ChatGPT Plugins[6].



Fig 2. AI Agents (red) will communicate with existing human written applications (purple) via a microservices architecture

Footnote Styling

[2] https://gizmodo.com/github-survey-92-percent-programmers-are-using-ai-tools-1850534570
[3] https://ai.googleblog.com/2022/07/ml-enhanced-code-completion-improves.html
[4] https://www.databricks.com/blog/introducing-english-new-programming-language-apache-spark
[5] https://thenewstack.io/how-ai-helped-us-add-vector-search-to-cassandra-in-6-weeks/
[6] https://openai.com/blog/chatgpt-plugins

The brain as a mental model for AI agent architecture

Because GenAI is most likely to be used to replace or enhance existing human behavior, it is important to use a new "mental model" for AI agents[7]. A natural consequence is that AI agents will likely be organized similar to the human brain, where there are separate modules that work in concert with each other.

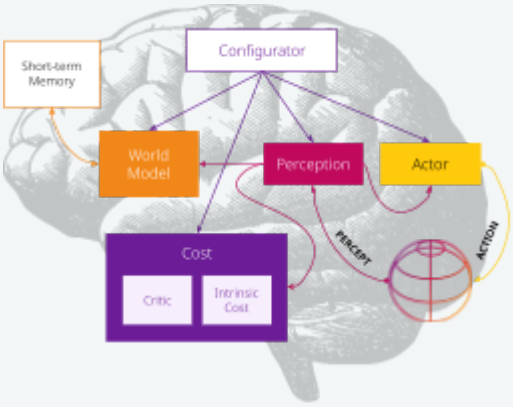


Fig 6. Fine-tuning sub-models of the multi-model agents

PRINCIPLE 1

Separation of memory from models

In the generative AI architecture, there are two leading ways to represent information: "implicitly" within the weights and biases of a transformer neural network, and "explicitly" as facts stored in a vector database. Retrieval augmented generation[8] (RAG) is a popular technique that is employed to retrieve context from a memory system to construct a prompt that is passed to a model. This approach results in fewer hallucinations and provides access to real-time information that is not encoded into the models.

[7] https://openreview.net/pdf?id=BZ5a1r-kVsf
[8] https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-foundation-models-customize-rag.html

PRINCIPLE 2

Decompose your business logic into “modules”

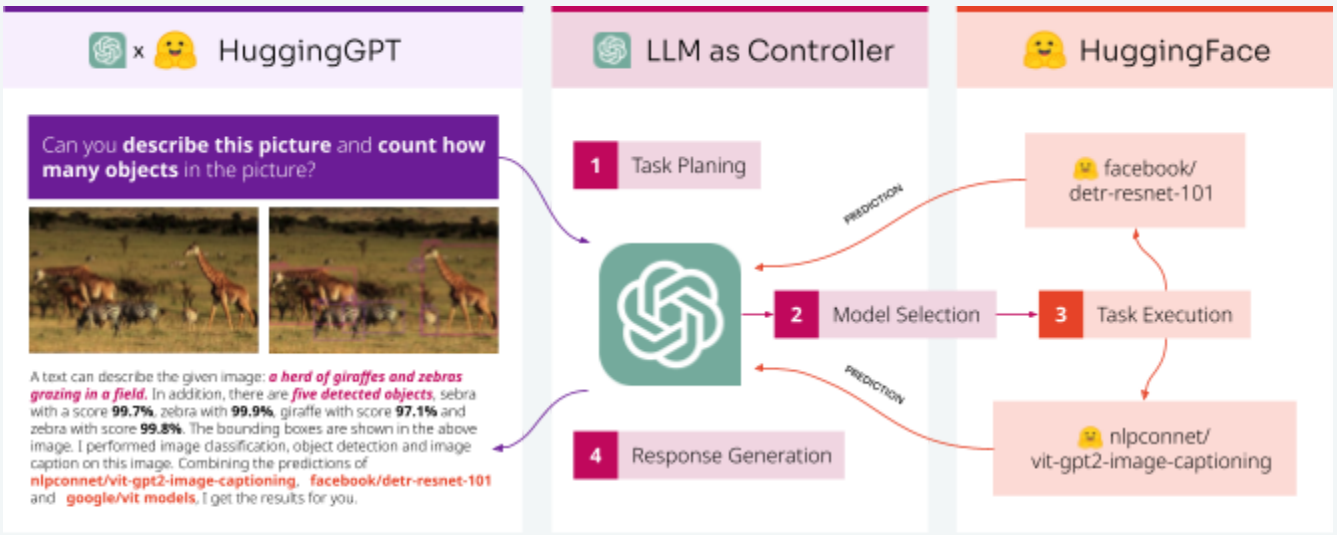


Figure 4: Language serves as an interface for LLMs (e.g., ChatGPT) to connect numerous AI models (e.g., those in Hugging Face) for solving complicated AI tasks. In this concept, an LLM acts as a controller, managing and organizing the cooperation of expert models. The LLM first plans a list of tasks based on the user request and then assigns expert models to each task. After the experts execute the tasks, the LLM collects the results and responds to the user

To get a computer to carry out complex tasks, it is a common practice to break down a system into several modules that interact with each other. Similarly, AI agents can consist of separate models for carrying out certain tasks.

This type of architecture was pioneered by Microsoft[9] (JARVIS), and is being adopted by LangChain's agent architecture[10]. In this architecture, the LLM itself acts as the configurator, and at run time, decides which AI models to invoke to accomplish a complex task.

The **configurator module** takes inputs from all other modules and configures them to perform the task at hand.

The **perception module** estimates the current state of the world. The perception module can be built from generative AI models or predictive models (e.g. CNN), or a combination of both.

The **world model module** predicts possible future world states as a function of imagined action sequences proposed by the actor. The world model module can be built with generative AI or predictive AI.

The **cost module** computes a single scalar output called "energy" that measures the effectiveness of the AI agent. The cost module takes the output of the world model, and determines how close to an objective. It isn't clear yet how good GenAI can be used for the cost module, but a combination of business rules (application logic) and causal reasoning is likely to suffice for many applications.

The **actor module** computes proposals for action sequences. The world model and the critic compute the possible resulting outcomes. The actor can find an optimal action sequence that minimizes the estimated future cost, and output the first action in the optimal sequence. This too can leverage GenAI to create an optimal action sequence.

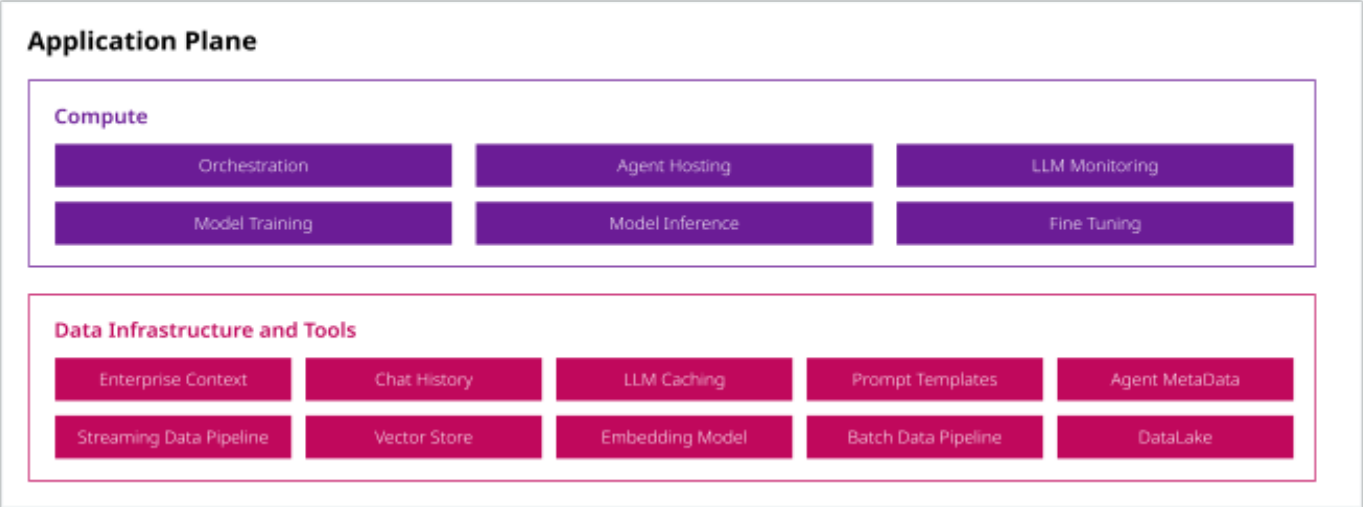
The **short-term memory module** keeps track of the current and predicted world states and associated intrinsic costs. Today, vector databases are the most popular way to store this information.

It is important to call out that the brain is a useful mental model for decomposing your system, but it should only be used as a mental model.

[9] https://arxiv.org/pdf/2303.17580.pdf
[10] https://docs.langchain.com/docs/components/agents/

The GenAI stack

The following diagram shows the GenAI stack. For brevity purposes, the stack does not include existing systems such as messaging platforms, SQL databases, etc (namely, components needed to run today's enterprise systems). The diagram also doesn't separate out the control plane from the application plane.



To understand how these various components interact with one another, it can best be broken up into five major workflows:



Model Training + Fine-Tuning

Real-time Data Processing / Embedding

Prompt Construction / Retrieval

Prompt execution / inference

Multi-Model & Multi-Agent Orchestration

5 major workflows



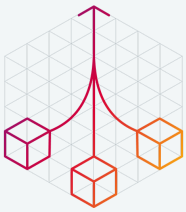
Model Training + Fine-Tuning: This stage consists of building foundation models, followed by fine-tuning the models with private data. The resulting model is then used for inference when the applications are running.



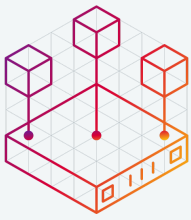
Data preprocessing / embedding: This stage involves storing private data (legal documents, in our example) to be retrieved later. Typically, the documents are broken into chunks, passed through an embedding model, then stored in a specialized database called a vector database.



Prompt construction / retrieval: When a user submits a query (a legal question, in this case), the application constructs a series of prompts to submit to the language model. A compiled prompt typically combines: a prompt template hard-coded by the developer; examples of valid outputs called few-shot examples; any necessary information retrieved from external APIs; and a set of relevant documents retrieved from the vector database.

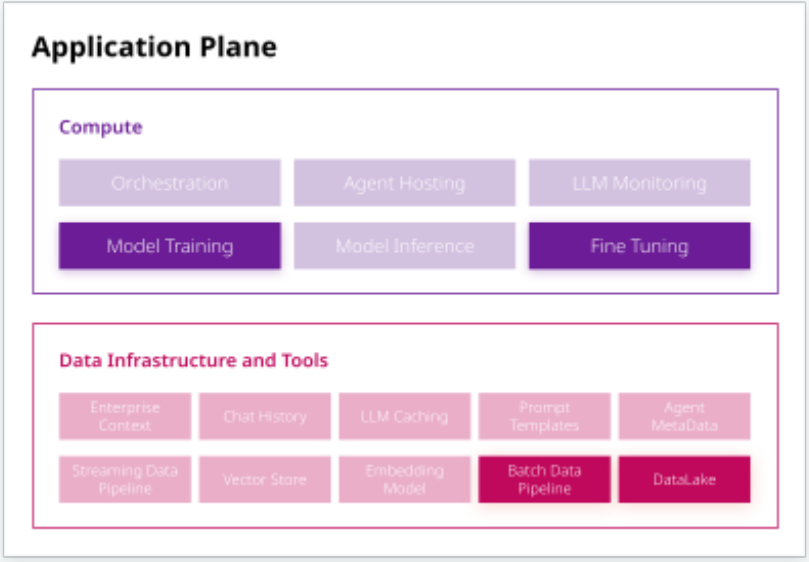


Prompt execution / inference: Once the prompts have been compiled, they are submitted to a pre-trained LLM for inference—including both proprietary-model APIs and open-source or self-trained models. Some developers also add operational systems like logging, caching, and validation at this stage.



Multi-Model / Multi-Agent Orchestration: More complex applications can consist of multiple modes/agents calling each other, similar to how applications today consist of smaller microservices that call each other.

Model training + fine-tuning



Foundation models are trained on a vast corpus of unstructured and structured data from across the internet.

Although it has been reported that today's foundation models take significant amounts of time to train, there is also another trend of very small foundation models being trained, too.

The smaller foundation models are likely much less powerful than the large ones being trained at companies like Google and OpenAI. The smaller foundation models are likely to still have the capabilities to understand language, but may lack the ability to have built in background knowledge. Regardless, the foundation model is still a key element of the overall compute stack.

Fine-tuning[11] models, also known as distillation[12], enables an enterprise to train an existing foundation model with their own data. The result is a much faster and smaller model that can be used for the enterprise. Contextual data for LLM apps includes text documents, PDFs, and even structured formats like CSV or SQL tables.

Instead of having a "one-giant-model," it is more likely that there will be a multitude of fine-tuned models in the enterprise, mimicking how our human brain works. The human brain works via a "Perception-Action" loop; one can expect specialized fine-tuned models to take the function of perception, action, simulation, and cost.

The question is then, how often will fine-tuning of these various models occur? More likely than not, the fine-tuning of these models will take place when there is a critical mass of information stored in the "memory" that is not implicitly stored in the weights and biases of the individual models. For human beings, our perception, action, world, cost models are updated usually nightly during our sleep.[13] It is likely that a similar schedule will occur for fine-tuning.

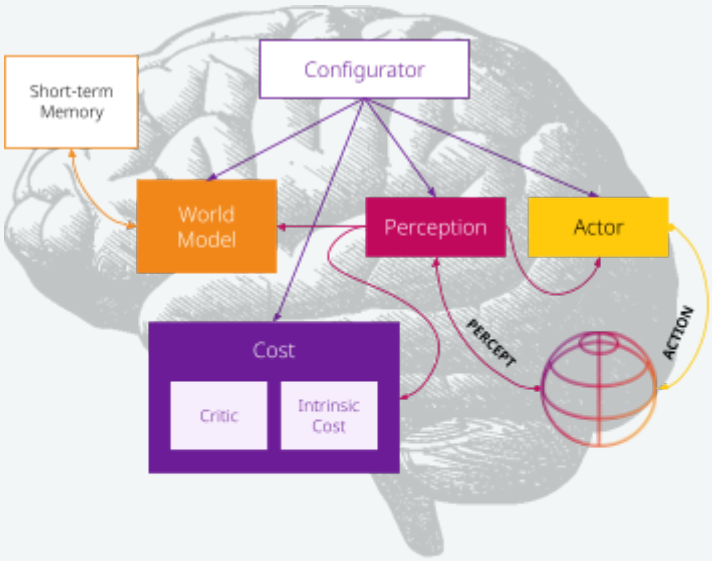
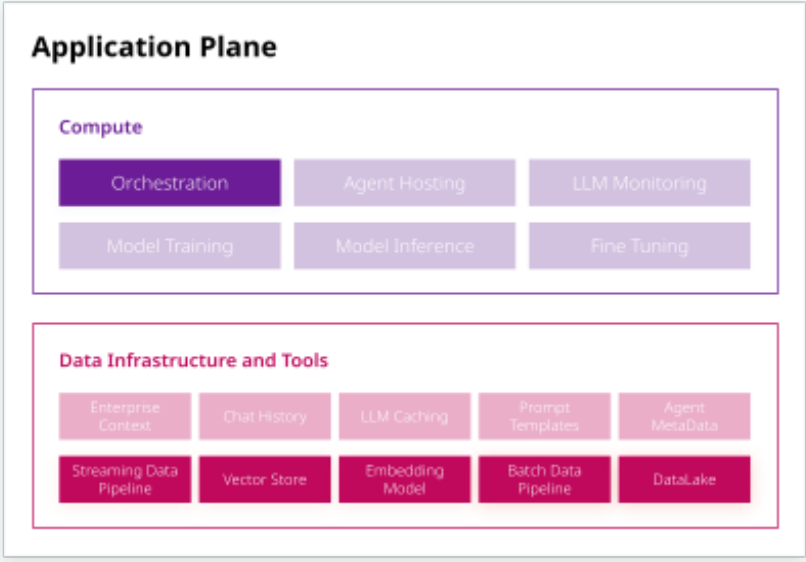


Fig 6. Fine-tuning sub-models of the multi-model agents

[11] https://en.wikipedia.org/wiki/Fine-tuning_(deep_learning)
[12] https://arxiv.org/abs/1910.01108
[13] https://www.deeplearningbook.org/

❯ Real-time data processing / embedding



Contextual data for LLM apps includes text documents, PDFs, and even structured formats like CSV or SQL tables.

Data-loading and transformation solutions for this data vary widely across developers we've spoken with.

Most use traditional ETL tools like Databricks or Airflow. Some also use document loaders built into orchestration frameworks like LangChain (powered by Unstructured) and LlamaIndex (powered by Llama Hub). We believe this piece of the stack is relatively underdeveloped, especially when it comes to real-time data ingestion at scale. DataStax is building a data ingestion system based on Apache Pulsar.

For embeddings, most developers use the OpenAI API, specifically with the text-embedding-ada-002 model. It's easy to use (especially if you're already already using other OpenAI APIs), gives reasonably good results, and is becoming increasingly inexpensive. Some larger enterprises are also exploring Cohere, which focuses its product efforts more narrowly on embeddings and has better performance in certain scenarios. For developers who prefer open-source, the Sentence Transformers library from Hugging Face is a standard. It's also possible to create different types of embeddings tailored to different use cases; this is a niche practice today but a promising area of research.

The most important piece of the preprocessing pipeline, from a systems standpoint, is the vector store. It's responsible for efficiently storing, comparing, and retrieving up to billions of embeddings (i.e., vectors).

There's a huge range of vector databases available, though. Notably:

- **Open source systems like Apache Cassandra, Elastic, Weaviate, Vespa, and Qdrant:** They generally provide excellent performance and can be tailored for specific applications, so they are popular with experienced AI teams who prefer to build bespoke platforms.
- **Local vector management libraries like Chroma and Faiss:** They offer a great developer experience and are easy to spin up for small apps and dev experiments. They don't necessarily substitute for a full database at scale.
- **OLTP extensions like pgvector:** For devs who see every database-shaped hole and try to insert Postgres—or enterprises who buy most of their data infrastructure from a single cloud provider—this is a good solution for vector support. It's not clear, in the long run, if it makes sense to tightly couple vector and scalar workloads.

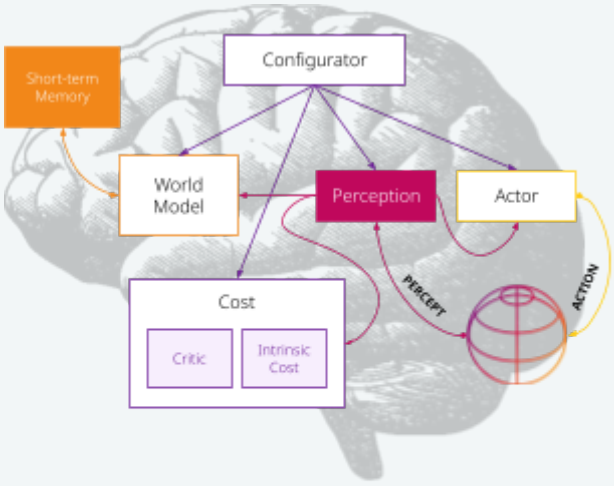
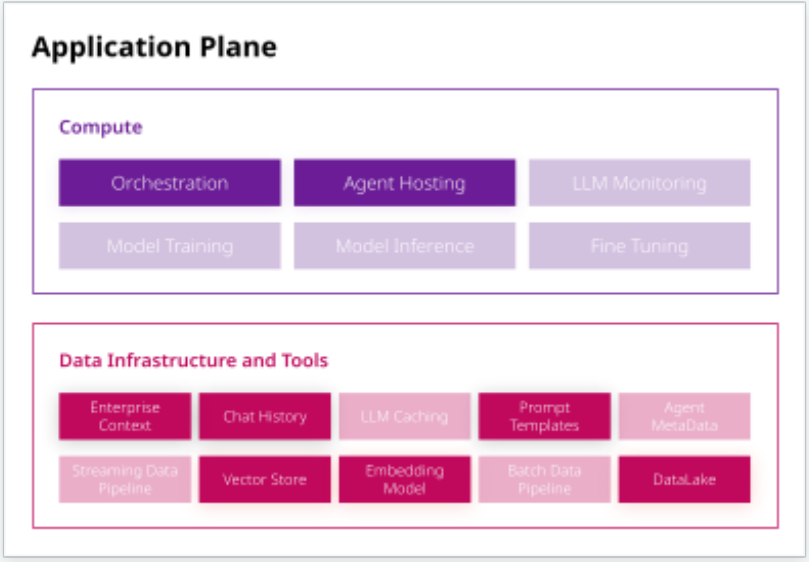


Fig 6. Data Processing / Embedding is the process of Perception interacting with Memory

Using the analogy of the brain, the data pipeline that processes outside information and stores it into memory can be seen as a **perception module** in our brain. In fact, the encoding of text/picture information into vectors for fast semantic retrieval mirrors how the brain encodes information.

The key requirement for vector databases is the ability to scale both reads and writes. Although it's used to persist data that appears to not change a lot, vector databases have a very large write requirement that we will see in the coming section.

Prompt (context) construction



Strategies for prompting LLMs and incorporating contextual data are becoming increasingly complex—and increasingly important as a source of product differentiation.

Most developers start new projects by experimenting with simple prompts, consisting of direct instructions (zero-shot prompting) or possibly some example outputs (few-shot prompting).

These prompts often give good results but fall short of accuracy levels required for production deployments. This is why RAG has become so important.

The key challenge of constructing a prompt from a vector store is to make sure that it is highly contextual, and fits within an LLM's token limit. Prompts are constructed from structured data sources such as SQL databases, as well as unstructured data sources such as vector databases or traditional search.

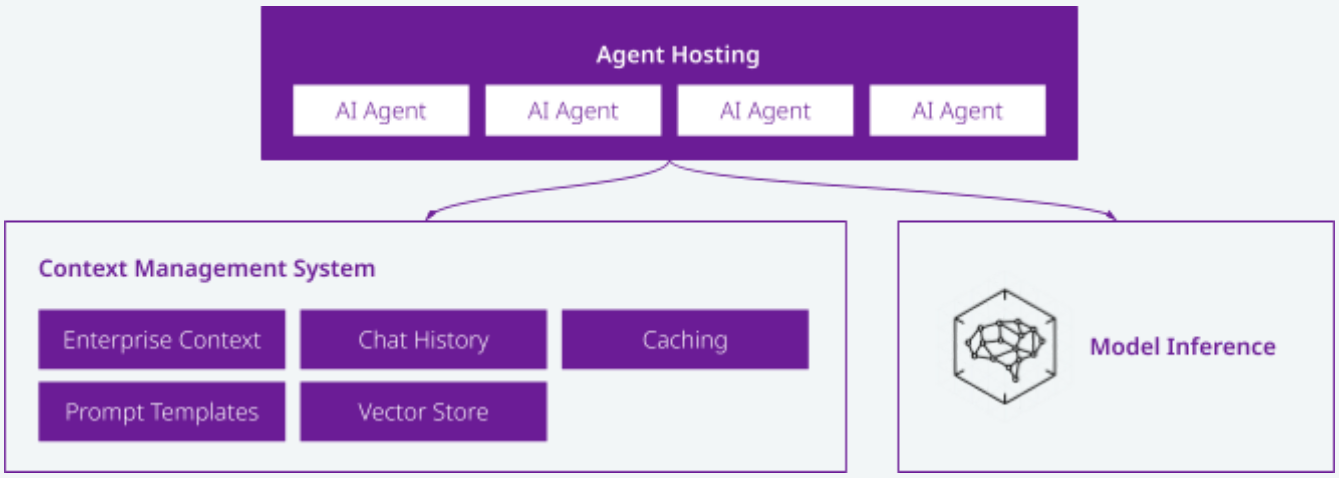
Because of the complexities of constructing a prompt, we expect a new class of abstraction to emerge: **context management systems**. Context management systems share many attributes with the more traditional content management systems for websites:

- They typically interact with multiple sources of data to build the context
- There are fairly sophisticated templating and authoring systems - aka prompt templates
- They must operate in real time

Context sources can loosely be broken into:

1. Context from private data sources **(enterprise context)**
2. Context from previous interactions with AI agents **(chat history)**

For each of the context sources, both vector store and SQL look-ups on data lakes / SQL OLTP DBs will be employed.



The actual hosting of the agent itself will be in an agent hosting environment. The agent itself is a lightweight application that takes care of constructing the prompt and invoking the LLM via a model inference service.

It is also important to recognize that we are in the infancy of LLMs in terms of the "context window" that they can support. Today, GPT3 supports 4,000 tokens; GPT-4 supports 8,000. Newer models from Claude support 100,000 tokens. As the token limit grows for the underlying LLMs, the level of sophistication needed for prompt construction also increases.

From a "brain as a mental model" perspective, being able to recall explicit information, and pass that information to a model that encodes implicit information, is a significantly more effective way to ensure that the answers coming from an LLM are grounded in facts (in other words, not a hallucination). Therefore, the ability to load facts from memory becomes a key ingredient in LLM performance.

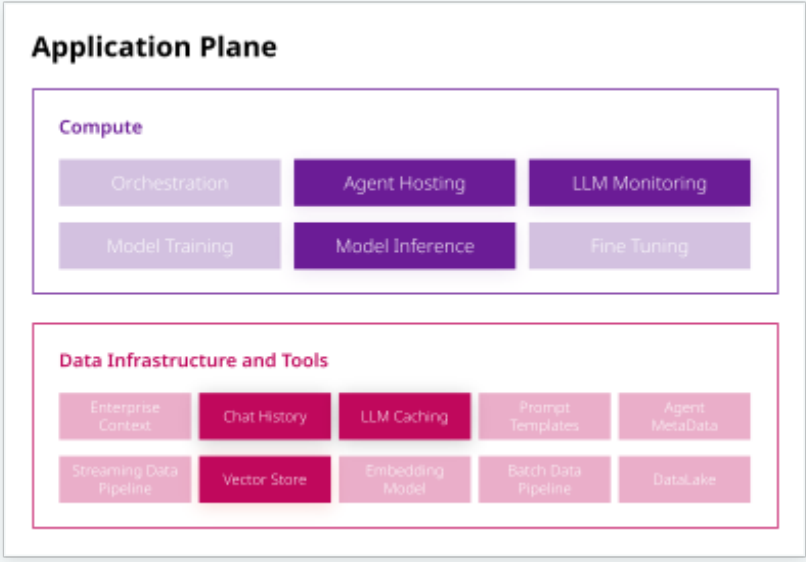
For SQL lookups, there are already many types of connectors that can translate natural language to SQL queries using LLMs, bypassing the need for developers to hand-create SQL. Similarly, advanced algorithms such as forward-looking active retrieval[14] use LLMs and vector search to take incoming questions, turn them into a variety of questions based on other questions raised from a knowledge base using an LLM, and re-queries the vector store for more information.

The last aspect of prompt construction is focusing the prompt to instruct an LLM on doing a very specific task. In the brain analogy, one can create prompt templates that are specifically designed for perception, actor, world model, and cost modules. So it's better to think of the prompt templates themselves as a "model," and the location of where the prompt templates are stored as a type of generative AI model registry[15].

[14] https://arxiv.org/abs/2305.06983

[15] https://www.phdata.io/blog/what-is-a-model-registry/

❯ Prompt execution / inference



Now that a prompt has been constructed, it needs to be executed on a LLM.

The AI agent responsible for invoking the LLM is running in an **agent hosting** system such as Vercel or AWS Functions. DataStax also has the ability to run "streaming agents" by running within a Pulsar environment.

Today, there are a number of LLM vendors that provide model inference services. The most popular are OpenAI and Google VertexAI. There are also many open source models such as LLama2 and Hugging Face.

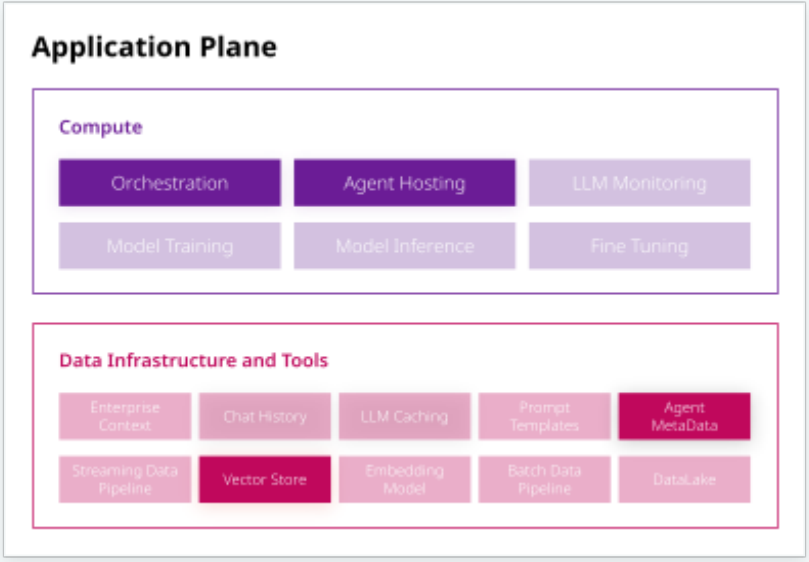
There are two major infrastructure concerns that CIOs must contend with:

1. **Model inference is expensive.** Given that all the code is effectively running on LLM models, it is not surprising that this cost will increase significantly.
2. **Storing the inference results is expensive.** GenAI generates massive amounts of data, and that data needs to be accessible in the future via the **chat history** that's backed by a **vector store**.

Because of these costs and scale, there is a large number of ancillary services that need to be invoked:

- **LLM Monitoring** - Similar to monitoring websites, one needs to monitor the chat conversations to determine accuracy and performance of the chats. However, because LLM is generating unstructured text, one really needs a separate system that uses natural-language processing techniques to determine if the answers are correct, further increasing the overall model inference costs
- Because model inference is so expensive, **LLM caching** is very important. However, because the input and output of an LLM is typically unstructured data, one would need a cache system that allows for "semantic" cache key lookups. Again, vector databases are the perfect system for performing LLM caching.

Multi-model and Multi-agent orchestration



Sophisticated AI Agents use multiple generative model invocation[16] to facilitate complex interactions.

The prompt engineering guide catalogs no fewer than 12 more advanced prompting strategies, including chain of thought, self-consistency, generated knowledge, tree of thought, directional stimulus, and others.

These strategies can also be used in conjunction to support different LLM use cases like document question answering, chatbots, etc. What these strategies have in common is that they are chaining together multiple LLM responses to generate a final interaction. The advantage of this approach is the following:

- LLMs typically perform much better when they are given very clear directives in their prompt.
- Because there is a limit to the context window of an LLM, you can provide an LLM more context by chaining multiple models. There is actually strong research evidence that long context windows for LLMs perform poorly[17], so it makes sense to keep the context one provides to an LLM as short as possible.

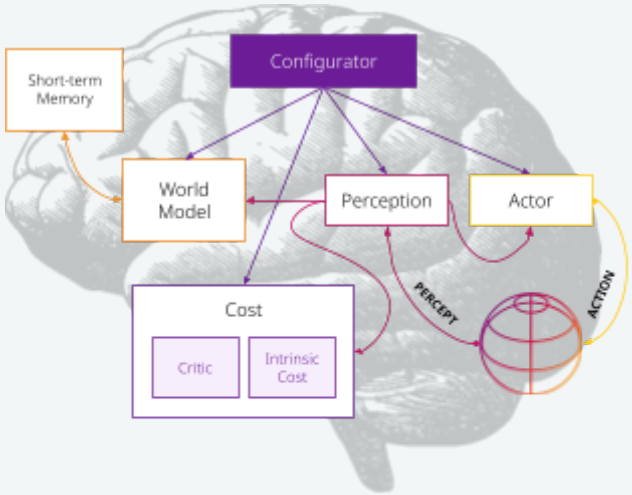


Fig 8. Configurator "wires up" multiple models to create an AI Agent

[16] https://medium.com/memory-leak/foundational-model-orchestration-fomo-a-primer-ba6c1464a82a
[17] https://tinyurl.com/cvpxvu26

- Because the LLMs are performing very specific roles - perception, actor, world model, and cost, they can be fine-tuned and significantly reduced in network size. This in turn reduces the net overall cost of model inference.
- Debugging becomes drastically easier, because each model only performs a certain task, and because the outputs of those tasks are stored in the "chat history," one can debug and determine which model is not working properly. Fixes such as modifying the prompt template can be made easily. One can literally "correct an agent's thinking" by using a multi-model approach.

This is where orchestration frameworks like LangChain and LlamaIndex shine. They abstract away many of the details of prompt chaining; interfacing with external APIs (including determining when an API call is needed); retrieving contextual data from vector databases; and maintaining memory across multiple LLM calls. They also provide templates for many of the common applications mentioned above. Their output is a prompt, or a series of prompts, to submit to a language model. These frameworks are widely used among hobbyists and startups looking to get an app off the ground, with LangChain being the leader. To store this information, an **agent metadata** abstraction will likely be necessary. Using the brain analogy, a **"configurator"** would read from an **agent metadata service**, and construct a directed acyclic graph of models to execute.

Finally, just like humans, agents themselves can be assembled into a set of collaborative AI agents. This is no different than enterprise applications being assembled into a set of services. In the paper, *Generative Agents: Interactive Simulacra of Human Behavior*[18], multiple AI agents collaborate to accomplish tasks like hosting a party. In a business scenario, it's been shown that if one asks a question of different agents with different personas (solo performance prompting), the answers that come back are significantly better[19].

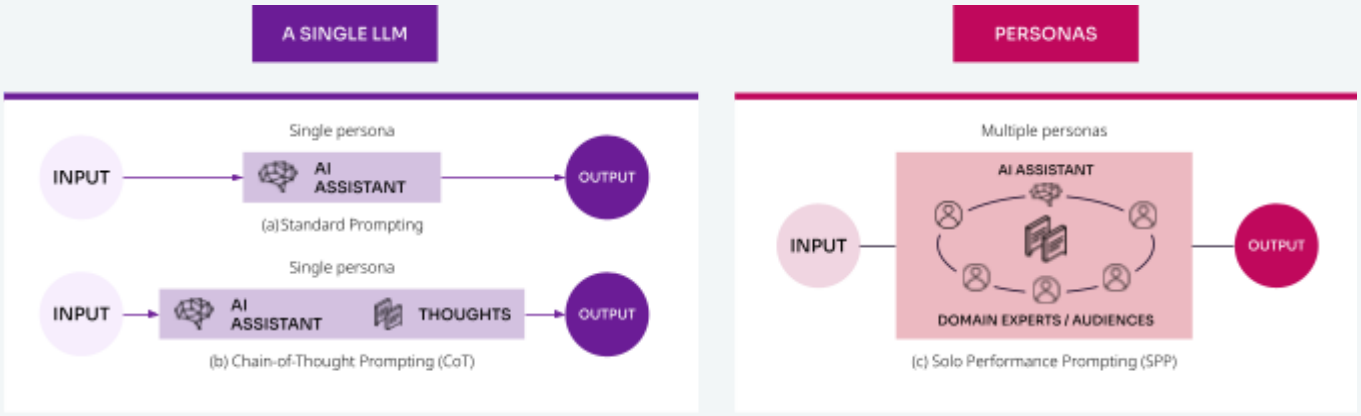


Fig. 9: Schematic illustration of solo performance prompting (SPP) and the difference compared to previous prompting methods. SPP transforms a single LLM into a cognitive synergist that dynamically identifies personas and engages in multi-turn self-collaboration to solve various tasks effectively.

[18] https://arxiv.org/abs/2304.03442
[19] https://arxiv.org/pdf/2307.05300.pdf

What's the gap between GenAI and human-level AI (AGI) ?

In Yann Lecunn's paper[20], A Path Towards Autonomous Machine Intelligence, it is argued that true human-level artificial intelligence can't be reached unless models are able to "reason" and "plan." Specifically, the "cost" module needs the ability to reason, and the "actor" module needs the ability to plan. Surprisingly, generative AI has the ability to plan[21] and reason[22] to some extent, making generative AI much more capable than AI researchers had expected. However, the inability to reason properly—especially because generative AI cannot truly understand "cause and effect"—makes it prone to certain issues such as bias or the inability to ensure its actions meet certain ethical guidelines. This is likely to be solved via another branch of AI called causal AI. Causal AI[23] at the moment does not scale to the needs of industry, but this is an area of intense focus. In the meantime, having human beings able to integrate into AI agents to perform the task of "reasoning" is necessary.'

[20] https://openreview.net/pdf?id=BZ5a1r-kVsf
[21] https://github.com/Significant-Gravitas/Auto-GPT
[22] https://arxiv.org/abs/2305.00050
[23] https://towardsdatascience.com/causal-discovery-6858f9af6dcb

Conclusion

GenAI is driving the evolution of the stack, transforming the application and control planes.

Illustrated through 5 different workflows, there are three key take-aways for the reader:

1. There will be a massive requirement to scale both model inference and vector stores
2. The GenAI stack will consist of abstractions and primitives that are drastically different from current day architectures
3. GenAI agents will more closely resemble how the human brain, and how humans interact.

The coming years promise to be transformative for all sectors, and as we make these bold steps forward, it's crucial that we're not just ready for change but that we're leading it. The revolution is here, and together, we can shape its course to maximize its potential and shape the future of humanity.

